# Short Answer:

Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail (Provide examples for each question).

1. What is data modeling? Why do we need it? When would you need it?

Data Modeling

● Data Model is the process of analyzing business requirement, designing and

creating physical instance(a plan or a blueprint) for the database or data

warehouse.

● It is a stage in SDLC(Software Development Life Cycle)

● It also impacts the user interface

1. What is an entity? What is an attribute? What is a tuple? What is a Domain?

● Entity

○ An item that can exist independently or

uniquely identified

● Attribute

○ Column label(name)

● Domain

○ Set of valid values for an attribute

● Relationship

○ How entities relate

● Degree

○ How many entities in a relationship

● Cardinality

○ Measure of participation

Graphical user interface

Description automatically generated

Table

Description automatically generated with low confidence

1. What is a super key? Give an example of a superkey that is not a candidate key and explain

why.

Table

Description automatically generated

A superkey is a set of attributes in a relation (table in a database) that can uniquely identify a tuple (row in a table). A candidate key is a minimal superkey, meaning it is the smallest set of attributes that can uniquely identify a tuple.

For example, consider a relation with the following attributes:

* Employee ID (integer)
* Name (string)
* Age (integer)
* Gender (string)

A superkey for this relation could be {Employee ID, Name, Age}, since this set of attributes can uniquely identify an employee. However, this is not a candidate key, because {Employee ID} is a smaller set of attributes that can also uniquely identify an employee. Therefore, {Employee ID} is a candidate key, and {Employee ID, Name, Age} is a superkey, but not a candidate key.

Another example of a superkey that is not a candidate key could be {Employee ID, Gender}. While this set of attributes can uniquely identify an employee, the smaller set {Employee ID} is also a unique identifier, and therefore {Employee ID} is the candidate key.

1. What is the primary key? How is it different from a unique key?

Primary Key vs Unique key

🡪 Remember the 2 Ns.

1st N NULLS :- Unique can have NULLS , but primary key can not have NULLS.

2nd N Numbers :- Many unique keys but only ONE Primary key.

Primary keys

A unique class instance

Foreign keys

Reference to another class

## **Key differences between Primary and Unique Key**

The following points explain the key differences between primary and candidate keys:

* A primary key can constitute one or more fields of a table to identify records in a table uniquely. On the other hand, a unique key prevents two rows from having duplicate entries in a column.
* A table cannot have more than one primary key in a relational database, while there can be multiple unique keys per table.
* A primary key column cannot contain NULL values, whereas a unique key can have NULL values, but only one NULL is allowed in a table.
* A primary key should be unique, but a unique key cannot necessarily be the primary key.
* The primary key by default is a clustered index where data is physically organized in the sequential index. In contrast, the unique key is a unique non-clustered index.
* The primary key implements entity integrity, whereas the unique key enforces unique data.

|  |  |  |
| --- | --- | --- |
| * **Comparison Basis** | **Primary Key** | **Unique Key** |
| **Basic** | The primary key is used as a unique identifier for each record in the table. | The unique key is also a unique identifier for records when the primary key is not present in the table. |
| **NULL** | We cannot store NULL values in the primary key column. | We can store NULL value in the unique key column, but only one NULL is allowed. |
| **Purpose** | It enforces entity integrity. | It enforces unique data. |
| **Index** | The primary key, by default, creates clustered index. | The unique key, by default, creates a non-clustered index. |
| **Number of Key** | Each table supports only one primary key. | A table can have more than one unique key. |
| **Value Modification** | We cannot change or delete the primary key values. | We can modify the unique key column values. |
| **Uses** | It is used to identify each record in the table. | It prevents storing duplicate entries in a column except for a NULL value. |
| **Syntax** | We can create a primary key column in the table using the below syntax:  CREATE TABLE Employee  (  Id INT PRIMARY KEY,  name VARCHAR(150),  address VARCHAR(250)  ) | We can create a unique key column in the table using the below syntax:  CREATE TABLE Person  (  Id INT UNIQUE,  name VARCHAR(150),  address VARCHAR(250)  ) |

<https://www.javatpoint.com/primary-key-vs-unique-key#:~:text=A%20primary%20key%20can%20constitute,multiple%20unique%20keys%20per%20table>.

A primary key is a column or set of columns in a relation (table in a database) that is used to uniquely identify each tuple (row in a table). It is a candidate key that is chosen to be the main means of identifying tuples in the table. The primary key must satisfy the following conditions:

* It must contain a unique value for each tuple in the table.
* It cannot contain a null value.

A unique key is similar to a primary key, in that it is a column or set of columns that is used to uniquely identify each tuple in a table. However, a unique key may contain a null value, whereas a primary key cannot. A table can have multiple unique keys, but it can only have one primary key.

For example, consider a relation with the following attributes:

* Employee ID (integer)
* Name (string)
* Age (integer)
* Gender (string)

The attribute {Employee ID} can be used as a primary key, since it is unique and cannot contain a null value. The attribute {Name} could be used as a unique key, since it is also unique, but it could contain a null value.

In summary, the primary key is a candidate key that is chosen to be the main identifier for tuples in a table, and it cannot contain a null value. A unique key is a column or set of columns that is used to uniquely identify tuples in a table, but it may contain a null value

Diagram

Description automatically generated

1. What are the relationships in data modeling?

● Cardinality is the number of times the entity participates in the relationship

○ One-to-One: One element in entityA may link to one element in entityB and vice versa

○ One-to-Many: One element in entityA may link to many elements in entity but one element in

entity may only link to one element in entityA

○ Many-to-One: Reverse A and B in one-to-many

○ Many-to-Many: One element in entityA may link to any number of elements in entity and vice

versa

● Degree

○ Degree is the number of entities involved in the relationship and it is usually 2(binary

relationship) however Unary and higher degree relationships can exists.

● Cardinality != Degree

Diagram

Description automatically generated Diagram

Description automatically generated

In data modeling, relationships refer to the connections between entities in a database. There are several types of relationships that can exist between entities:

1. One-to-one relationships: In this type of relationship, one entity is related to only one instance of the other entity. For example, a driver's license can be related to one person.
2. One-to-many relationships: In this type of relationship, one entity is related to multiple instances of the other entity. For example, a teacher can be related to multiple students.
3. Many-to-many relationships: In this type of relationship, multiple instances of one entity are related to multiple instances of the other entity. For example, a student can be enrolled in multiple courses, and a course can have multiple students enrolled in it.
4. Self-referencing relationships: In this type of relationship, an entity is related to itself. For example, an employee can be related to their manager, who is also an employee.

These relationships are typically represented in a data model using lines connecting the entities. The type of relationship is indicated by the symbols used at the ends of the lines. For example, a line with a single arrowhead pointing at an entity indicates a one-to-many relationship, while a line with a double arrowhead indicating a many-to-many relationship.

1. What is cardinality in data modeling? What are the different types of cardinalities?

Give an example for each type.

Cardinality in data modeling refers to the number of occurrences of one entity that can be related to a single occurrence of another entity. There are three types of cardinality:

1. One-to-one cardinality: In this type of cardinality, one occurrence of one entity is related to only one occurrence of the other entity. For example, a driver's license can be related to one person.
2. One-to-many cardinality: In this type of cardinality, one occurrence of one entity is related to multiple occurrences of the other entity. For example, a teacher can be related to multiple students.
3. Many-to-many cardinality: In this type of cardinality, multiple occurrences of one entity are related to multiple occurrences of the other entity. For example, a student can be enrolled in multiple courses, and a course can have multiple students enrolled in it.

It's important to note that cardinality is different from the types of relationships that can exist between entities. Cardinality refers specifically to the number of occurrences of one entity that can be related to a single occurrence of the other entity, while relationships refer to the connections between entities in general.

1. What are composite attributes, multi-valued attributes, and derived attributes? Give at least

one example for each type of attribute that is not in the course material.

1. **Composite attribute ---- ii) Attribute that can be further subdivided to yield additional attributes**
2. **Composite attribute -------- iii) Attribute that can have more values**
3. **Derived Attribute --------  i) Attribute whose value is calculated from other attributes**

Composite attributes are attributes that are made up of multiple smaller attributes. For example, a "full name" attribute could be a composite attribute made up of a "first name" attribute and a "last name" attribute.

Multi-valued attributes are attributes that can have multiple values for a single entity. For example, a person entity could have a "hobbies" attribute that is multi-valued, allowing the person to have multiple hobbies such as painting, gardening, and cooking.

Derived attributes are attributes that can be calculated or derived from other attributes. For example, a "total cost" attribute for an order could be derived by multiplying the quantity of each item by its price.

Examples of composite attributes that are not in the course material:

* A "full address" attribute could be a composite attribute made up of a "street address," "city," "state," and "zip code" attribute.
* A "employee ID" attribute could be a composite attribute made up of a "department code" attribute and a "employee number" attribute.

Examples of multi-valued attributes that are not in the course material:

* An "email addresses" attribute for a person entity could be multi-valued, allowing a person to have multiple email addresses.
* A "certifications" attribute for an employee entity could be multi-valued, allowing an employee to have multiple certifications.

Examples of derived attributes that are not in the course material:

* A "discounted price" attribute for a product could be derived by applying a discount percentage to the original price attribute.
* A "total distance" attribute for a car rental could be derived by multiplying the number of days the car was rented by the average number of miles driven per day.

1. How do you represent a many-to-many relationship in a database? Please describe in

detail using an example.

A many-to-many relationship in a database can be represented using a junction table. A junction table is a separate table that contains foreign keys to the two entities that are involved in the many-to-many relationship.

For example, consider a database of students and courses. Each student can be enrolled in multiple courses, and each course can have multiple students enrolled in it. This represents a many-to-many relationship between the student and course entities.

To represent this relationship in a database, we would create a junction table called "enrollments" with foreign keys to the student and course tables. The enrollments table would contain a row for each enrollment, with the student's ID and the course's ID.

Here is an example of how the tables might look:

## Students table: ID | Name

1 | Alice 2 | Bob 3 | Eve

## Courses table: ID | Name

1 | Math 2 | English 3 | History

## Enrollments table: StudentID | CourseID

1 | 1 1 | 3 2 | 2 3 | 1 3 | 2 3 | 3

In this example, the enrollments table allows us to represent the many-to-many relationship between students and courses. Alice is enrolled in Math and History, Bob is enrolled in English, and Eve is enrolled in Math, English, and History.

ERD

● ER-Diagram

○ Entity Relationship Diagram

○ Used to create or design a blueprint of the Database

or Data Warehouse

○ Design Entities, attributes and show relationships

Diagram

Description automatically generated

Crow’s Foot Notation

● Connection Symbols display Relationships

● Entity and Attributes in Table like format

Diagram

Description automatically generated Table

Description automatically generated Table

Description automatically generated

Convert ERD to Tables

● Each entity type becomes a table

● Each single-valued attribute becomes a column

● Derived attributes are ignored/computed column

● Multi-valued attributes are represented by a separate table

● Use Conjunction Table to break up the Many-to-Many relationship

● The key attribute of the entity type becomes the primary key or unique key of the

table Diagram

Description automatically generated

1. What is normalization? Why do you need to normalize?

● Main goal of normalization

● Reduce redundancy, avoid anomaly and create a well-structured series of table

without error or inconsistencies

● Minimize redesign when extending the database structure

○ New type of data can be accommodated without changing existing structure too much

● Ensure data dependencies are properly enforced by data integrity

constraints(Entity Integrity, Referential Integrity, Domain Integrity)

● Anomaly异常

● When an attempt is made to modify(update, insert into, or delete from) a

relation, the following undesirable side-effects may arise in relations that have

not been sufficiently normalized

● Anomaly is the issue that may occur because of redundancy

● Types: Update, Insertion and Deletion

● Anomaly Update

● The same information can be expressed on multiple rows; Therefore, updates to

the relation may result in logical inconsistencies

Table

Description automatically generated with low confidence

● Anomaly Insertion

● There are circumstances in which certain facts cannot be recorded at all Table

Description automatically generated

Normalization is a process of scaling data so that it has values between 0 and 1. It is a common technique that is applied to datasets before they are used for machine learning.

There are several reasons why normalization is important:

1. Machine learning algorithms often assume that the input data is in a certain range, and if the data is not in that range, the algorithms may not work as well. Normalization helps to ensure that the data is in the correct range.
2. Normalization can help to reduce the impact of outliers in the data. Outliers are extreme values that can have a large impact on the results of machine learning algorithms. Normalizing the data can help to reduce the influence of these outliers.
3. Normalization can make the training process faster and more stable. If the data is not normalized, the training process may take longer and be less stable.
4. Normalization can improve the interpretability of the results. If the data is not normalized, it may be difficult to understand the results of the machine learning algorithms because the scale of the data may be misleading. Normalization can help to make the results more interpretable.

There are several different ways to normalize data, including min-max normalization and standardization. It is important to choose the appropriate normalization method for your specific dataset and machine learning problem.

Normalization is a database design technique to remove redundant data.

Normalization is implemented by splitting tables in to two, one with reference data ( master table) and other transaction data.

 Denormalization is a database design technique to improve search performance. In denormalization we merge tables so that we need to fetch from less tables and thus increase search performance.

**1st normal form :-** A table is in first normal form when the columns have Atomic values. It should not have repeating groups.

**2nd Normal form :-** First normal form should be satisfied. All non-key columns should be fully dependent on the Primary key.

**3rd Normal :-** All 1st and 2nd normal forms should be satisfied. No transitive dependency should be present.

1. What does data redundancy mean? How is it different from duplication? Can you give an

example of each?

● Redundancy

○ Values repeated unnecessarily in multiple records or fields within one or more tables

Table

Description automatically generated

Data redundancy refers to the presence of multiple copies of the same data within a database. This can occur when data is stored in multiple tables or when multiple copies of the same data are stored in a single table.

Duplication, on the other hand, refers to the presence of identical copies of the same data within a database. This can occur when the same data is entered multiple times, either by accident or due to a lack of proper data management.

An example of data redundancy would be a database that stores customer information in two separate tables: one table for customer names and contact information, and another table for customer purchase history. In this case, the customer's name and contact information would be stored in both tables, resulting in data redundancy.

An example of duplication would be a database that stores customer information in a single table, but the same customer's name and contact information is entered multiple times. This could happen if the database is not properly designed to prevent duplicate entries, or if the data is entered manually and there are errors in the process.

Data redundancy and duplication can both cause problems in a database, including wasting storage space, increasing the risk of data inconsistencies, and making it more difficult to manage and analyze the data. It is important to design and maintain databases in a way that minimizes redundancy and duplication.

1. What are the different types of dependencies? How are they different? Give an example

of each type.

● Functional Dependencies

○ Functional Dependencies are how different

attributes relate in a table

○ At this level, we focus on individual tables

○ We see how individual attributes relate to

the keys in the table

■ Primary Key & Candidate Keys =

Prime Attributes

■ Attributes that aren’t keys = NonPrime Attributes

● Types of Dependencies

○ Full Dependencies – Depends on all prime

attributes fully

○ Partial Dependencies – Depends on some

Prime Attributes

○ Transitive Dependencies – Depends on an

attribute that depends on a Prime Attribute

1. What are normal forms? Which normal forms are most common?

● First Normal Form

○ Each table cell should contain a single value

○ Each record needs to be unique

● Second Normal Form

○ Meets all of 1NF

○ Makes sure all non-prime attributes are fully dependent on a prime attribute

● Third Normal Form

○ Meets 1NF and 2NF

○ Every non-prime attribute is non-transitively dependent on the prime attributes

Box and whisker chart

Description automatically generated with medium confidence Diagram

Description automatically generated

Normal forms are a set of rules for organizing data in a database. They are designed to minimize data redundancy and improve the integrity and efficiency of the database. There are several different normal forms, ranging from first normal form (1NF) to fifth normal form (5NF).

The most common normal forms are first normal form (1NF) and third normal form (3NF).

1NF is the most basic normal form and requires that all data in a database be stored in a tabular format with a unique identifier for each row (called a primary key). It also requires that each column in the table contain a single value and that there be no repeating groups of data.

3NF is a more advanced normal form that builds upon 1NF. It requires that all non-primary key columns in a table be directly dependent on the primary key and that there be no transitive dependencies (i.e., no non-primary key columns that depend on other non-primary key columns).

Other normal forms, such as second normal form (2NF) and fourth normal form (4NF), are less commonly used in practice, but may be applied in certain situations to further optimize the design of a database.

It is important to design a database according to the appropriate normal form to ensure that it is efficient, accurate, and easy to maintain.

1. What is database integrity? Why do you need it? Provide an example of user-defined

integrity.

● Entity Integrity

○ Design of the table or entity

○ String PK with no nulls or repeats

● User-Defined Integrity

○ Rules or constraints applied by the user to maintain rules of design

● Domain Integrity

○ Correct and proper domains specified with proper use of columns

● Referential Integrity

○ Proper FK setup with proper PK reference

○ Good design for connection and joins

○ Data integrity is crucial especially because there us a lot of user input

Database integrity refers to the accuracy and consistency of data in a database. It is important to maintain database integrity because it ensures that the data is reliable and can be used for the intended purposes.

There are several types of integrity that can be enforced in a database, including:

1. Entity integrity: Ensures that each row in a table has a unique identifier (primary key).
2. Referential integrity: Ensures that foreign keys in a table refer to existing primary keys in another table.
3. Domain integrity: Ensures that data in a column is of the correct data type and falls within a specified range or set of values.
4. User-defined integrity: Custom rules that are defined by the database administrator or user to enforce specific business rules or constraints.

An example of user-defined integrity is a rule that ensures that all dates in a table are in the past. This rule could be implemented using a check constraint, which is a type of user-defined integrity that allows the user to specify a condition that must be met for data to be accepted into the database. In this case, the check constraint would ensure that all dates in the table are less than the current date.

Maintaining database integrity is important because it helps to ensure the accuracy and reliability of the data, which is essential for making informed decisions and achieving the desired results from the database.

1. What is a relational database? What is a non-relational database? Difference?

When would

you choose a relational or non-relational database?

Relational Database vs. Non-Relational Database

● Relational Database(SQL)

○ Traditional way of storing data

○ Data is stored in tables with rows and columns

○ Rigid schema with well-defined relationships

○ Difficult to scale

○ Examples: SQL Server, Oracle, MySQL

● Non-Relational Database(NoSQL)

○ Developed more recently

○ Data can be stored in a variety of formats(JSON documents, key-value pairs, wide-column,

graphs)

○ Flexible schema with loose relationships

○ Easily scalable

○ Examples: MongoDB, Redis

A relational database is a type of database that stores data in the form of tables, with each table consisting of rows (called records) and columns (called fields). The tables are related to each other through common fields (called keys), which allows data to be linked and queried in a logical and organized way.

A non-relational database, also known as a NoSQL database, is a type of database that does not use the traditional table-based relational model. Instead, it stores data in a format that is more flexible and scalable, such as documents, key-value pairs, or graphs.

There are several key differences between relational and non-relational databases:

1. Data structure: Relational databases use a structured data model, where data is organized into tables with fixed columns and rows. Non-relational databases use a more flexible data model, where data can be stored in a variety of formats and does not have to adhere to a fixed structure.
2. Scalability: Non-relational databases are typically more scalable than relational databases, as they can handle large volumes of data and high levels of traffic without requiring complex data modeling.
3. Performance: Non-relational databases are generally faster than relational databases, as they do not require the overhead of maintaining relationships between tables.
4. Data integrity: Relational databases generally provide stronger data integrity, as they enforce rules such as primary keys, foreign keys, and check constraints to ensure the accuracy and consistency of the data. Non-relational databases may not provide the same level of data integrity, but they may offer other benefits such as flexibility and scalability.

Which type of database to choose depends on the specific requirements of the application. Relational databases are a good choice for applications that require a structured and consistent data model, such as financial systems or customer relationship management systems. Non-relational databases are a good choice for applications that require flexibility and scalability, such as social media platforms or online shopping websites.

Top of Form

1. What is DDL? What are the major statements in DDL?

● DDL – Data Definition Language ● Create ● Alter ● Drop ● Truncate Text

Description automatically generated

● This command builds a new table and has a predefined syntax

Text

Description automatically generated

● The mandatory semicolon at the end of the statement is used to process every command before it. In this example, the string CHAR is used to specify the data type. Other data types can be DATE, INT, DECIMAL etc.

Alter

● An Alter command modifies an existing database table. This command can add

up additional column, drop existing columns and even change the data type of

columns involved in a database table.

Graphical user interface, application, Teams

Description automatically generated

● In this example, we added a unique primary key to the table to add a constraint

and enforce a unique value. The constraint “Product\_Id” is a primary key and is

on the Products table.

Drop

● A drop command is used to delete objects such as a table, index or view. A

DROP statement cannot be rolled back, so once an object is destroyed, there’s

no way to recover it.

Graphical user interface, text, application

Description automatically generated

Truncate

● Similar to DROP, the TRUNCATE statement is used to quickly remove all

records from a table. However, unlike DROP that completely destroys a table,

TRUNCATE preserves it’s full structure to be reused later.

Graphical user interface, text, application, chat or text message

Description automatically generated

1. What is DML? What are the major statements in DML?

● DML – Data Manipulation Language ● Insert ● Update ● Delete

● Insert Statements

○ Insert statements are used to input data into tables

○ The order of data specified should match order of columns

○ If you don’t know the order of columns, specify each column name in the insert statement

● For example

○ Inserting data matching order

■ Insert into TableName values(1, ‘Name’),(2, ‘Name’);

○ Inserting data without knowing matching order

■ Insert into TableName(Col2, Col1) values(‘Name’, 1),(‘Name’, 2);

● Update Statements

○ Update statements are used to change or modify data inside a table

○ Specify single row depending on statement

○ Use unique identifiers to get correct rows

● For Example

○ Update using unique column

■ Update TableName Set Name = ‘Tim’ Where ID = 2;

○ Update using non-unique column

■ Update TableName Set Name = ‘Hal’ Where Name = ‘Bob’

● Delete Statements

○ Delete statements are used to remove specific rows inside a table

○ Use unique values to identify the correct rows to delete

○ Delete leave logs and continue identity values

● For Example

○ Deleting rows using specific unique values

■ Delete from TableName Where ID = 1;

○ Deleting rows using non-unique values

■ Delete From TableName Where Name = ‘Jim’

1. How do you insert values into a table if you don’t know the order of the columns?

There are several ways to insert values into a table if you don't know the order of the columns:

1. Specify the column names in the INSERT statement: You can use the INSERT statement to specify the column names and corresponding values that you want to insert into the table. For example:

INSERT INTO my\_table (column1, column2, column3)

VALUES (value1, value2, value3);

1. Use the DEFAULT keyword: If you don't have a value for a particular column, you can use the DEFAULT keyword to insert the default value for that column. For example:

INSERT INTO my\_table (column1, column2, column3)

VALUES (value1, DEFAULT, value3);

1. Use a SELECT statement with the default values: You can use a SELECT statement to insert the default values for all columns in the table. For example:

INSERT INTO my\_table

SELECT DEFAULT, DEFAULT, DEFAULT FROM dual;

1. Use the INSERT ALL statement: You can use the INSERT ALL statement to insert multiple rows into the table at once, and specify the values for each column in each row. For example:

INSERT ALL

INTO my\_table (column1, column2, column3) VALUES (value1, value2, value3)

INTO my\_table (column1, column2, column3) VALUES (value4, value5, value6)

INTO my\_table (column1, column2, column3) VALUES (value7, value8, value9)

SELECT \* FROM dual;

1. How is truncate different from delete?

Truncate deletes all rows from a table while Delete deletes specific rows. Truncate is DDL and Delete is DML. Truncate command is faster than Delete command. Truncate resets identity while Delete retains the identity.

Truncate and delete are two SQL commands that are used to remove data from a database table. However, there are several key differences between the two:

1. Performance: Truncate is generally faster than delete, as it does not need to scan the table for rows to delete and does not generate any undo or redo logs.
2. Rollback: Truncate cannot be rolled back, as it permanently removes the data from the table. Delete can be rolled back using a transaction, as it only marks the rows as deleted and the data remains in the table until the transaction is committed.
3. Triggers: Truncate does not activate any delete triggers on the table, as it does not delete the rows one by one. Delete activates the delete triggers for each row that is deleted.
4. Referential integrity: Truncate may violate referential integrity constraints, as it does not check for foreign key dependencies before deleting the data. Delete does not violate referential integrity constraints, as it only deletes rows that do not have foreign key dependencies.

In general, truncate is used when you want to quickly and permanently remove all the data from a table, while delete is used when you want to selectively remove specific rows from a table. It is important to choose the appropriate command based on your specific needs and the constraints of the database.

1. Is syntax in SQL case sensitive?

SQL is not case sensitive, which means that you can use upper or lower case letters to write SQL commands and they will be treated the same. However, some databases may have specific rules for the case of table and column names, and you may need to use the correct case when referring to these objects in your SQL statements.

For example, in some databases, table and column names are stored in lower case and must be written in lower case in your SQL statements. In other databases, table and column names are stored in upper case and must be written in upper case in your SQL statements.

It is important to check the documentation for your specific database to determine the rules for case sensitivity in table and column names. In general, it is a good practice to be consistent with the case of your SQL statements and use the same case as the names of the tables and columns in your database.

1. What is the order of different clauses in a select statement when you write it?

● DQL – Data Query Language

● Select From Where

○ Select: Pick up which column of data you’d like to fetch

○ From: Select which table or data set to fetch.

○ Where: Specific a criteria to sort data by use operators(Filter).

■ Operators: In, Or, And

● Group by, Having, Order by

○ Group by – Used to combine similar values in column

○ Having – filter conditions for aggregate only

○ Order by – display the data by order by a specific column Text

Description automatically generated Table

Description automatically generated

The order of the clauses in a SELECT statement is as follows:

1. SELECT: This clause specifies the columns that you want to retrieve from the database.
2. FROM: This clause specifies the tables that you want to retrieve data from.
3. WHERE: This clause specifies a condition that must be met for rows to be included in the results.
4. GROUP BY: This clause groups the results by one or more columns.
5. HAVING: This clause specifies a condition that must be met for groups to be included in the results.
6. ORDER BY: This clause sorts the results by one or more columns.

Here is an example of a SELECT statement with all of the clauses:

SELECT column1, column2, AVG(column3)

FROM my\_table

WHERE column4 > 10

GROUP BY column1, column2

HAVING AVG(column3) > 20

ORDER BY column2 ASC;

Note that the WHERE, GROUP BY, HAVING, and ORDER BY clauses are optional and may not be included in every SELECT statement. It is important to choose the appropriate clauses based on your specific needs and the requirements of the query.

1. What are constraints? Why do we need them? Are they mandatory to have?

● A constraint is usually associated with a table and is created with a CREATE

CONSTRAINT SQL statement.

● They define certain properties that data in a database must comply with

● Key Constraints

○ Primary Key

■ 1 per table

■ Unique Clustered index

■ Not Null

○ Unique Key

■ 999 per table

■ Unique Non-Clustered index

■ 1 Null Allowed

○ Foreign Key

■ Cannot exist before PK

■ Must be deleted before PK

● Other Constraints

○ Null, Not Null

■ Are nulls allowed

○ Check

■ Data must meet rule

○ Default

■ If nothing, then this

○ Data types

■ Char(2) –States (NY, CA…)

■ Varchar(10) – Names…

■ Money – Mone

Constraints are rules that are enforced by the database to ensure the accuracy and integrity of the data. They are used to specify the allowed values, data types, and relationships between data in a database table.

There are several types of constraints that can be used in a database, including:

1. Primary key: A primary key is a unique identifier for each row in a table. It cannot contain null values and must be unique across all rows in the table.
2. Foreign key: A foreign key is a column that refers to the primary key of another table. It is used to establish a relationship between the two tables and ensure referential integrity.
3. Not null: A not null constraint specifies that a column cannot contain null values.
4. Check: A check constraint specifies a condition that must be met for data to be accepted into a column.
5. Unique: A unique constraint specifies that the values in a column must be unique across all rows in the table.

Constraints are important because they help to ensure the accuracy and consistency of the data in a database. They can prevent errors and inconsistencies from occurring, and ensure that the data is in a valid and predictable state.

Constraints are not mandatory to have in a database, but they are often used to enforce business rules and ensure the integrity of the data. It is a good practice to use constraints to ensure the quality and reliability of the data in your database.

1. How do you make sure there are no duplicate values in a column?

To ensure that there are no duplicate values in a column, you can use a unique constraint. A unique constraint specifies that the values in a column must be unique across all rows in the table.

To create a unique constraint, you can use the following syntax:

ALTER TABLE my\_table

ADD CONSTRAINT constraint\_name UNIQUE (column\_name);

This will prevent duplicate values from being inserted into the column. If an attempt is made to insert a duplicate value, the database will return an error and the insertion will be rolled back.

You can also use the SELECT DISTINCT statement to retrieve unique values from a column. This statement will return only the unique values from the column, and any duplicates will be removed from the results. For example:

SELECT DISTINCT column\_name FROM my\_table;

Using a unique constraint or the SELECT DISTINCT statement can help to ensure that there are no duplicate values in a column and maintain the integrity of the data in your database.

1. How many ways can you add constraints to a table? How are they different?

There are several ways to add constraints to a table in a database:

1. When creating the table: You can specify the constraints when you create the table using the CREATE TABLE statement. For example:

CREATE TABLE my\_table (

id INTEGER PRIMARY KEY,

name VARCHAR(50) NOT NULL,

date\_of\_birth DATE CHECK (date\_of\_birth < CURRENT\_DATE),

FOREIGN KEY (id) REFERENCES another\_table (id)

);

1. After creating the table: You can add constraints to an existing table using the ALTER TABLE statement. For example:

ALTER TABLE my\_table

ADD CONSTRAINT constraint\_name UNIQUE (column\_name);

1. Using the SET CONSTRAINTS command: You can use the SET CONSTRAINTS command to enable or disable all constraints on a table or in the entire database. For example:

SET CONSTRAINTS ALL DEFERRED; -- Disable all constraints

SET CONSTRAINTS ALL IMMEDIATE; -- Enable all constraints

1. Using the CREATE CONSTRAINT TRIGGER command: You can use the CREATE CONSTRAINT TRIGGER command to create a trigger that enforces a constraint on a table. Triggers are special procedures that are executed automatically when a specific event occurs in the database. For example:

CREATE CONSTRAINT TRIGGER trigger\_name

AFTER INSERT OR UPDATE ON my\_table

FOR EACH ROW

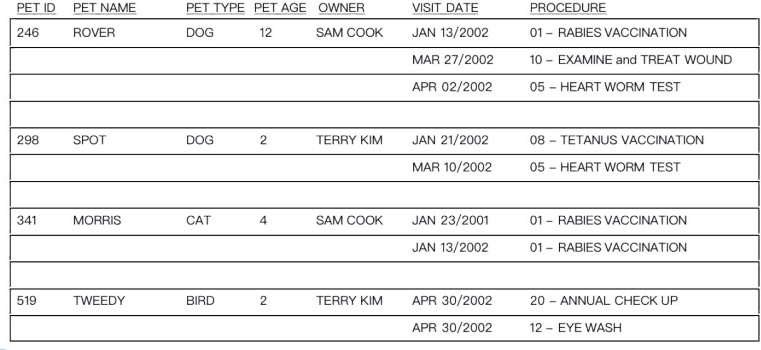
EXECUTE PROCEDURE constraint\_procedure();

Each of these methods has its own benefits and limitations, and it is important to choose the appropriate method based on your specific needs and the constraints of the database.

# Coding Questions:

Write code in c# to solve the following problems. Please write your own answers. You are highly encouraged to present more than one way to answer the questions. Please follow best practices when you write the code so that it is easily readable, maintainable, and efficient. Clearly state your assumptions if you have any. You may discuss with others on the questions, but please write your own code.

1. Please show the tables in their 1NF, 2NF, and 3NF.



1. We would like to design a database to maintain information about hospital staff,

including doctors and nurses, and patients at the hospital. The information we need includes

* 1. Staff, including their names, addresses, and social security numbers.
  2. Patients, including their names, addresses, and the name of their insurance company.
  3. Patients are each assigned to a ward (room).
  4. The staff that are nurses are assigned to zero or more wards. Each ward has at least one nurse assigned.
  5. The staff that are doctors are assigned to zero or more patients. Patients may or may not have a doctor assigned, and they may have more than one doctor.
  6. Patients in the same ward may have different doctors but will always have the same nurse(s). Please design the ERD.

1. Read the README file from adventureworks. And create the database in SSMS.
2. Return all columns and data regarding employees only
3. Return all the names of customer in adventureworks (all 3 columns)
4. List all customers whose first name is Eugene.
5. According to your answer to question 1, create tables. Populate at least three tables with data. Make sure to use different methods to populate tables. Write a query to delete all Owners whose first name is ‘Sam’.

\*Submit all your SQL files including those used for inserting data.